REINFORCEMENT LEARNING BASED ACTIVE LOCALIZATION FOR PRECISE MANIPULATION

A Thesis
Presented to
The Academic Faculty

By

Ayush Shukla

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in the
School of Electrical and Computer Engineering

Georgia Institute of Technology

May 2020

REINFORCEMENT LEARNING BASED ACTIVE LOCALIZATION FOR PRECISE MANIPULATION

Approved by:

Prof. Cedric Pradalier, Advisor
School of Interactive Computng
Georgia Institute of Technology

Prof. Paul Voss
School of Electrical and Computer Engineering
Georgia Institute of Technology

Prof. Samuel Coogan
School of Electrical and Computer Engineering
Georgia Institute of Technology

Hermann Blum
Autonomous Systems Lab
ETH Zurich

Dr. Abel Gawel
Autonomous Systems Lab
ETH Zurich

Date Approved: January 31, 2020

"Flirting with madness was one thing; when madness started flirting back, it was time to call the whole thing off."

– *Rohinton Mistry (A Fine Balance)*

To my family, for being there.

ACKNOWLEDGEMENTS

Foremost, I would like to thank my supervisors at ETH Zurich, Hermann Blum and Dr. Abel Gawel, for their accommodating and considerate spirit throughout the duration of the project. Thanks for the numerous healthy discussions, and for reorienting my work whenever it got too incoherent. Further, I'm grateful to Prof. Dr. Roland Siegwart and all the other members of the Autonomous Systems Lab at ETH Zurich for giving me the opportunity to spend a wonderful time working over there. I'll forever cherish the memories of my time in Switzerland.

I'd also like to express my sincere gratitude to my advisor, Prof. Cedric Pradalier, for his guidance, not just through this thesis, but throughout my whole Masters. I'm greatly indebted to you for your teaching and mentorship.

Most importantly, I'd like to acknowledge the unconditional love and support that I received from my family. Thanks to my parents, Drs. Arvind and Shailaja Shukla, for their constant encouragement, and for tolerating my, at times, irrational behaviour. Lastly, I thank my elder brother, Samarth, for letting me to stay at his house.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Precision during positioning of a robot arm is inherently limited by the sensing capabilities of the robot. In the case of fully on-board sensing, this limitation is further exacerbated by inconsistencies in the expected and actual structure of the robot environment. Further, precision is also affected by uncertainty in the estimate of the robot pose. While these sources of errors can never be fully eliminated, we can at least devise an optimal measurement policy for a given sensor configuration and robot environment. That is exactly what we try to do in this thesis. Such a policy would ensure that measurements are taken from places which lead to an optimal increase the localisation precision.

For our purpose, we assume that the robot end-effector is equipped with a sensor setup consisting of an array of highly accurate 1D laser rangefinders (pointlasers). Pointlasers provide a cheaper and more accurate alternative to a 3D LiDAR, at the expense of measurements which are sparsely distributed in the rotation space.

We treat this policy estimation problem as an active localization problem and set up a 3D simulation environment consisting of the CAD model of a building. Inside this environment, we simulate different orientations of the pointlaser array and train a reinforcement learning (RL) agent which predicts the best orientation for a given position in the CAD model. Using reduction in uncertainty (information gain) as the reward, we come up with a policy which takes reliable measurements and thus, should lead to precise positioning of the end-effector. We train the agent using an on-policy RL algorithm and present the results on a number of test CAD models.

CHAPTER 1
INTRODUCTION

1.1 Motivation

The key motivation for this work stems from a desire to automate tasks in the construction industry using mobile robots. A lot of these tasks are repetitive and dangerous for human workers. Further, constant changes in health regulations on construction sites render many traditional techniques infeasible. In spite of this, construction remains one of the least digitalised industries. Hence, there is a need for developing robots capable of performing construction tasks autonomously without human intervention. These tasks encompass fabrication tasks such as brick laying, as well as auxiliary tasks such as drilling, chiselling and direct fastening.

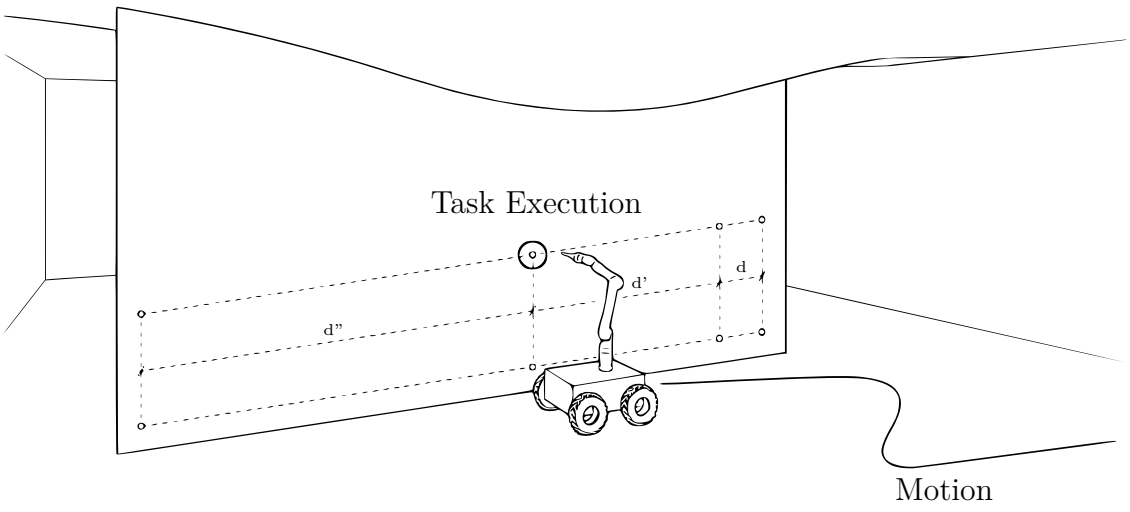


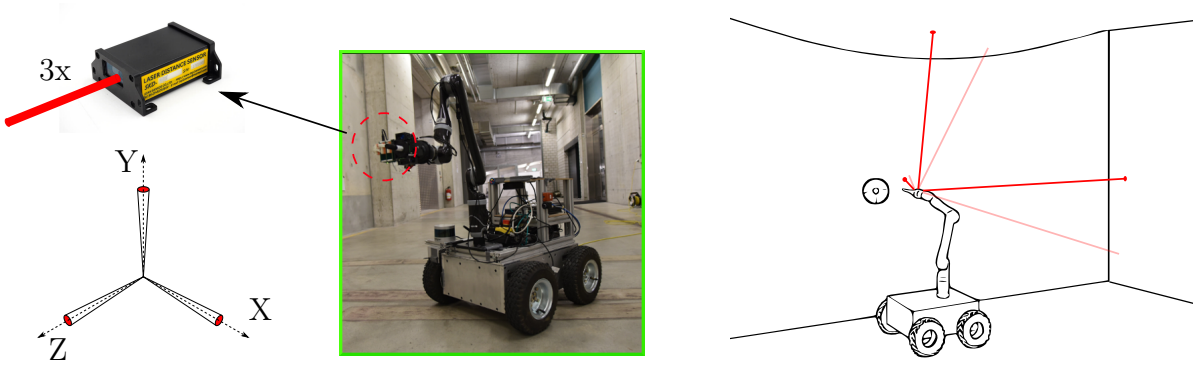
Figure 1.1: A mobile manipulator positioning it's end-effector at a target position

We limit our discussion to performing the auxiliary construction tasks discussed above. These tasks can be broadly split into two phases: movement of the person/machine to the task location, and subsequent placement of the construction tool precisely at the target

position. Precision is necessary since small deviations in the positioning of the tool could lead to a failure of the whole task. For example, in a drilling task, minute deviations in a set of holes may add up to an overall arrangement where the holes don't align with the object to be attached. Mobile manipulator robots, which have a manipulator arm mounted on a mobile platform (Figure 1.1), are ideal candidates for performing these tasks since they possess the required mobility and dexterity. Consequently, we hope to use one such robot, Waco (Figure 1.2a) to showcase this potential.

1.2 Objectives

As mentioned, we focus on the latter part of a construction task pertaining to high precision movement of the tool. Accordingly, we assume that the robot already has the ability to safely navigate the construction environment and roughly arrive at the task location. Moreover, we restrict ourselves to only on-board sensing for high accuracy localisation to ensure that the proposed system is flexible and can be easily adapted to different environments. Our sensor setup consists of an orthogonal array of three highly accurate pointlasers attached to the end-effector of the robot arm (Figure 1.2a). The overall objective, therefore, is to develop a sensing strategy (eg. Figure 1.2b) which is able to achieve high precision positioning of the end-effector using this sensor setup and a CAD model of the environment as a reference.



(a) Sensor configuration on the Waco robot [1] (b) Example of successive measurements

Figure 1.2: A visualisation of the project set-up

1.3 Approach

Unfortunately, a conventional decoupled approach to planning and subsequent sensing for this setup could lead to end-effector poses at which laser measurements taken are not reliable. This could be caused by poor references such as corners/gaps in the environment or references which are too far away. These unreliable measurements can lead to inaccuracies in the estimated end-effector position. Therefore, we propose an active sensing strategy which maximises an information gain metric of the localisation routine at each stage ('active localization'). This strategy is trained in simulation for a particular CAD environment using deep reinforcement learning (RL). The trained policy would constrain motion of the arm to poses from which measurements taken are reliable and hence, should lead to precise localisation of the end-effector.

We assume that there is an initial uncertainty of several centimetres in the end-effector position arising from a coarse global SLAM routine. This uncertain position is represented using a multivariate normal distribution. Initialising from this uncertain state, we train an RL agent to successively take measurements which result in maximal uncertainty reduction.

Further, instead of limiting ourself to the orthogonal array of three pointlasers installed on the actual robot, we tackle a generalised problem of RL based active localisation for a generic array of pointlasers.

1.4 Contributions

In summary, we demonstrate the following original contributions through this research:

- A new method for modelling measurement uncertainty for pointlasers.
- Deep Reinforcement Learning based active sensing strategy in pointlaser setups.

CHAPTER 2
LITERATURE REVIEW

2.1 Construction Robotics

Using mobile robots for performing construction tasks, especially digital fabrication, is an active area of research [2, 3]. Ardiny et al. [2] present an extensive review of the current research and challenges in this field. They conclude that the ability of mobile robots to perform construction tasks is hindered by the precision of current self-positioning systems. Precision is traditionally achieved by placing external markers in the environment like fiducial markers [4, 5] or external laser trackers [6, 7, 8]. In contrast, Dorfler et al. [9] demonstrate a completely autonomous brick stacking robot employing only an on-board 2D laser scanner attached to the end-effector. By moving the arm in a sweeping fashion around the task location, they first generate an initial reference 3D pointcloud of the environment using 2D laser scans. During the stacking phase, they align the current scan of the environment to the reference scan and localise within it. While their method achieves sub-cm accuracy sufficient for digital fabrication, tasks under our consideration (drilling, chiselling, direct fastening etc.) need a system which is more precise.

In a prior work [1], our group came up with a sensor setup consisting of an orthogonal array of three highly precise inexpensive 1D pointlasers attached to the end-effector. They demonstrated the capability of this sensor setup to achieve high accuracy localisation needed for construction tasks. Specifically, they used a heuristic-based localisation routine which compared the real laser measurements from six different hand-crafted poses around the task location to the expected measurements inferred from the building CAD model. By feeding these pairs of measurements into an optimiser, they were able to accurately localise the end-effector in the building CAD model. In our work, we plan to replace this hand-

designed strategy of taking measurements by an informative planning routine which takes into consideration the uncertainty involved in taking measurements from different poses.

2.2 Active Localization

In a conventional navigation routine, the motion planning stage does not take into account the sensing capabilities of the robot. The planner computes a motion plan by minimising only the costs (distance travelled, time taken etc.) associated with the movement. Online use of this strategy for movement involves an implicit passive localisation routine and thus, they are appropriate for sensor setups which provide dense and unambiguous measurements. Active Localization [10], on the other hand, couples the planning and sensing process to devise a navigation strategy which jointly maximises the information gain (minimising uncertainty), while minimising the associated movement costs. Mathematically, it tries to optimise the following criterion:

$$J = \min_{policy} \left\{ \sum_{j} \alpha_j \mathcal{U}_j + \sum_{l} \beta_l \mathcal{C}_l \right\}$$

where $\mathcal{U}_j$ is the expected uncertainty and $\mathcal{C}_l$ is the movement cost. The solution to the above optimisation problem is a sequence of motions (actions), often called a policy. In a probabilistic setting where measurements are noisy, this problem can be considered as a Partially Observable Markov Decision Process (POMDP). Such problems can be solved using value iteration methods (dynamic programming) [11, 12] or state based search methods [13].

In order to formulate our problem as an active localisation problem, we need to explicitly characterise a scalar information content (or uncertainty) metric for the robot and sensor setup. This can be done in two ways [10]: either using a scalar function of the covariance matrix of the belief distribution [13], or using an entropy-based function of the belief distribution [14].

Once formulated, the active localisation problem can be solved in multiple different ways

[15]. One way is to precompute the information content map of the whole state space which can then be used to find optimal actions at each state [14]. For a static environment, this is a one-time operation. The downside is that this approach has large memory requirements, especially for a high-dimensional state space. Alternatively, the information content at different states can be computed online in a lazy fashion, which could be then used for planning. There is a trade-off between memory and time between the two approaches.

Active sensing (localisation) has been shown to be useful in global mobile robot localisation problems [11, 12, 16]. In these problems, the robot has no prior knowledge of its position in the map and it takes movements which optimally allow it to localise itself. Value iteration on a sampled belief state is used to solve the optimisation. More recently, deep reinforcement learning approaches too have been used to tackle the global active localisation problem [17, 18]. These approaches train a separate perceptual model (measurement likelihood function) and an optimal policy model (distribution over possible actions) in an end-to-end fashion. We take a moment to emphasise the difference between these studies and our proposed research. In our scenario, we already have a good prior on the end-effector location and hence, our localisation routine is better described as local, rather than global. We mainly focus on using this prior and the environment CAD model to plan a trajectory which results in maximum information gain, and thereby, low uncertainty in the final position. Our proposed approach may also be referred to as informative path planning [19].

In the computer vision community, active sensing using cameras has been studied as an 'active vision' problem [20]. Here, the objective is to orient the camera towards locations which give a more informative view of the environment, relevant to the underlying task. This approach has been used for performing tasks such as inspection, reconstruction [19] and reliable localisation [21]. More recently, Zhang et al. [13] demonstrated an active vision based method to build a 3D discretised information map of the environment which can be efficiently queried to compute the Fisher Information metric for arbitrary 6D poses.

Of late, deep neural networks have been proposed to create this uncertainty map of

environment using 2D laser scan data [22, 23]. Since our sensor setup consists of sparse, though highly precise 1D pointlaser measurements, we instead try to estimate the information content using Monte Carlo sampling in simulation.

Deep reinforcement learning has lately been used to solve a diverse set of robotics problems [24, 25]. The end-to-end learning based approach alleviates the need to formulate and solve complex model dynamics involved in robotics.

To the best of our knowledge, we are the first to formulate the problem of precise 3D positioning using pointlaser data, as an active localisation problem, and to propose a deep RL based solution.

CHAPTER 3

SIMULATION ENVIRONMENT

For simulating pointlaser measurements inside a building, we've created a simple simulator on top of the Visualization ToolKit (VTK) library[1]. The building model is imported into VTK as a mesh extracted from a CAD file. The mesh is assumed to be water-tight i.e. it consists of a single closed surface [26]. VTK provides convenient and efficient ray-casting APIs which are used for simulating laser beams and their intersections with the building model (Figure 3.1).

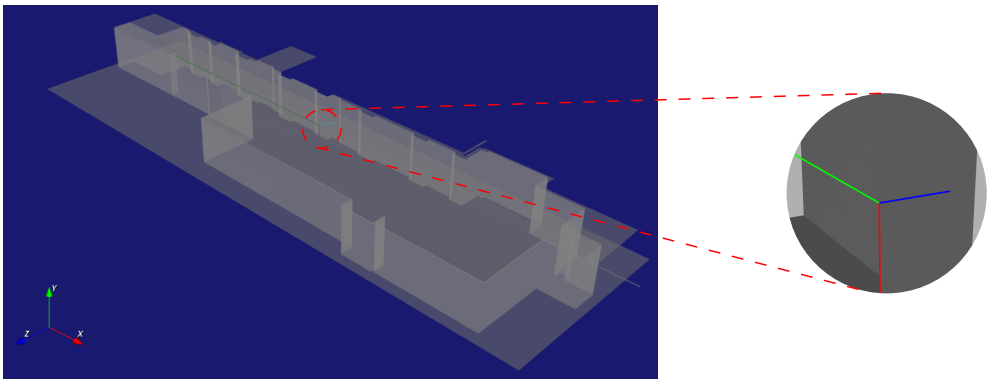


Figure 3.1: Pointlaser simulation inside a building CAD model in VTK

3.1 Robot State Distribution (Belief)

To limit the dimensionality of the problem, we model the robot state as a belief over only the position of the pointlaser array (and therefore, the end-effector). Here, we assume that a good enough estimate of the orientation is available at every instant. Also, the uncertainty in the orientation is assumed to be captured within the measurement model (as described in Section 3.2). This restricts our task to only estimating the position with high precision, ignoring the orientation. Further, since our localisation task is restricted to a local subspace

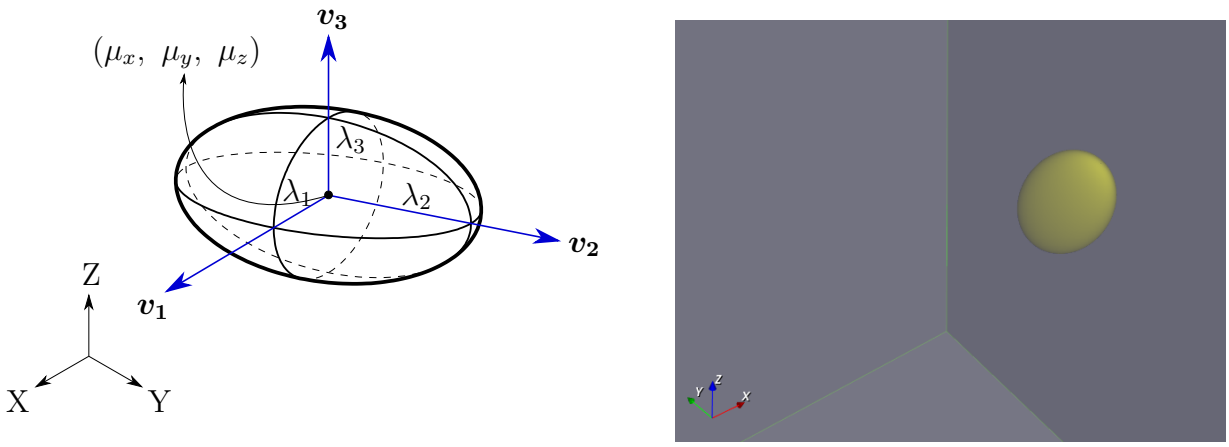
[1] `https://vtk.org/`

around an initial position estimate, it is reasonable to model the state belief as a local distribution. Hence, the state X is represented as a trivariate gaussian distribution of the x, y and z positions. This also has the advantage of being easily parameterised by a mean μ and covariance Σ.

$$X = (\mu,\ \Sigma)$$

where $\mu = [\mu_x, \mu_y, \mu_z]$ and $\Sigma = \begin{bmatrix} \Sigma_{xx} & \Sigma_{xy} & \Sigma_{xz} \\ \Sigma_{xy} & \Sigma_{yy} & \Sigma_{yz} \\ \Sigma_{xz} & \Sigma_{yz} & \Sigma_{zz} \end{bmatrix}$.

This state is visualised in VTK as an ellipsoid. The centre of the ellipsoid is the mean μ. The eigenvalues $\lambda_i s$ of the covariance Σ represent the lengths of the principal axes, while it's eigenvectors $\boldsymbol{v_i} s$ represent the ellipsoid orientation (Figure 3.2).



(a) Parameters of the ellipsoid

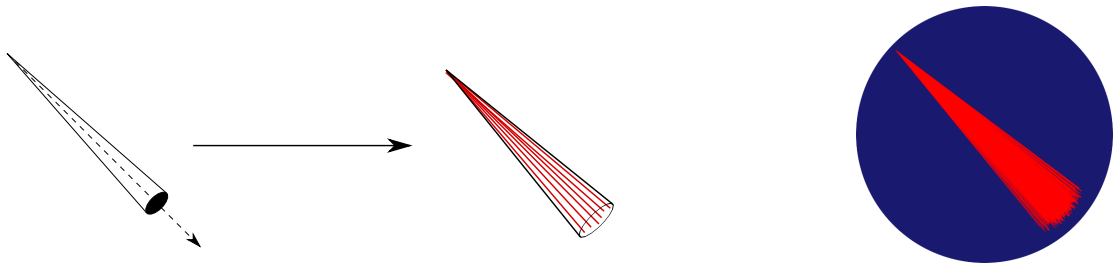
(b) VTK visualisation of a sample state

Figure 3.2: Representation of the state X distribution as an ellipsoid

Initially, the state covariance Σ is taken to be a diagonal matrix with large values representing large uncertainty from a coarse global localisation routine. As we take successive pointlaser measurements, the values in the covariance matrix decrease and we become more certain of the position.

3.2 Conical Sensor Model

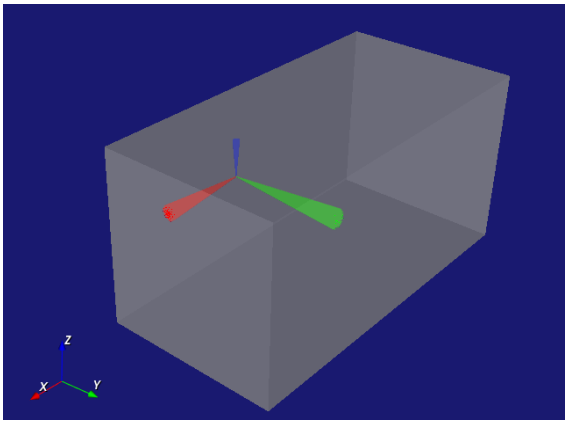
The purpose of a sensor model (likelihood model) is to provide a probability distribution of the possible measurements from a given pose. As mentioned earlier, we want this model to capture the uncertainty in the orientation since we assume that the estimate of the robot arm orientation is always noisy. We, therefore, model the pointlaser beams as having a uniform conical distribution around a central direction (Figure 3.3a). This is implicitly captured by assuming that the quaternion associated with the orientation of the pointlaser array has a conical distribution (refer to Appendix A for details).



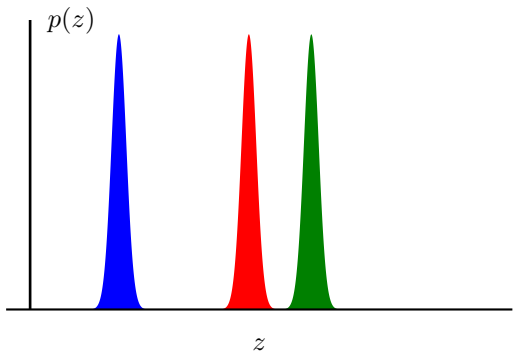
(a) Uniformly distributed laser beams inside cone (b) VTK simulation

Figure 3.3: Monte Carlo sampling of rays to approximate pointlaser measurement model

Now, in order to estimate the possible measurements and their associated likelihoods using this conical sensor model, we do Monte Carlo sampling over the possible orientations. We sample multiple orientations of the pointlaser array, perform measurements from these orientations, and accumulate the resulting values in separate histograms for each pointlaser (Figure 3.4). Ray-casting in VTK is used to get the points of intersection of the laser rays with the CAD model, and hence, the distance measurement. The measurements are presumed to be within a finite range between zero and a maximum laser distance value. This range is divided into equisized histogram bins and each sample contributes a count of one to the corresponding bin. Out of range measurements are assumed to contribute uniformly to all the histogram bins. Moreover, for simulating the inherent gaussian noise in each pointlaser ray measurement, we later convolve the histograms with a 1D gaussian filter having a standard deviation comparable to that of the real pointlaser.



(a) Conical ray samples from pointlasers

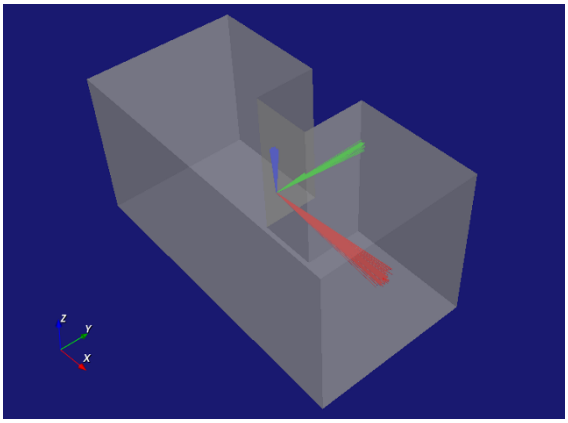


(b) The associated histogram likelihoods

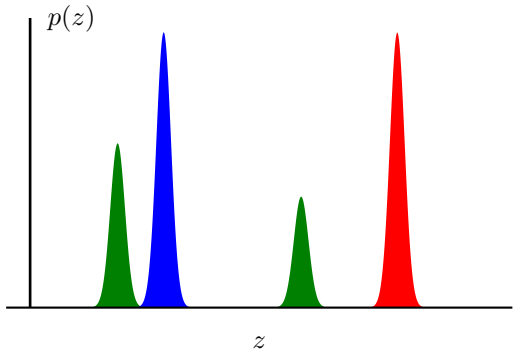
Figure 3.4: Example of measurement likelihoods from a point inside a test mesh

3.2.1 Monte Carlo sampling vs parametrically modelling the likelihood

As shown in Appendix B, even for a simple 2D case, the measurement noise is a nonlinear function of the angle of the pointlaser beam. A more complicated scenario would involve walls which are not flat and orientations at which the measurements have discontinuities (Figure 3.5). Since modelling such scenarios would become increasingly complex, we instead take a sampling approach to approximate the likelihood. Additionally, fitting parametric distributions like a gaussian to the sampled measurements is not necessary since histograms can be used to represent arbitrary distributions at the cost of memory usage.



(a) Conical ray samples from pointlasers



(b) The associated histogram likelihoods

Figure 3.5: Example of measurement likelihoods at a discontinuity

3.3 State Update

In the beginning, we assume a ground truth position of the pointlaser array which is sampled from the initial state distribution X_0. Subsequently, the state X_t at any instant is updated whenever a new measurement is taken from the ground truth position, following an action a_t. The action a_t is defined as a new absolute orientation of the pointlaser array. From this ground truth measurement, we update the prior belief X_{t-1} through Monte Carlo sampling over it's distribution. To elaborate, we take a random sample of points $\boldsymbol{x_i}$s, and calculate the likelihood of the measurement from each of these points. This is analogous to the update step of a particle filter. Using the calculated likelihoods as the weighted contributions of each point towards a posterior belief, we fit another trivariate gaussian as the new state, $X_{new} = (\mu_{new},\ \Sigma_{new})$. The weights are defined as

$$w_i = \prod_{lasers} p(z_{laser_j} \mid \boldsymbol{x_i}, a_t)$$

where z_{laser_j} is the ground truth measurement of the pointlaser j and $p(.)$ is its associated likelihood obtained from the histogram.

The mean of the new state μ_{new} is the weighted sample mean

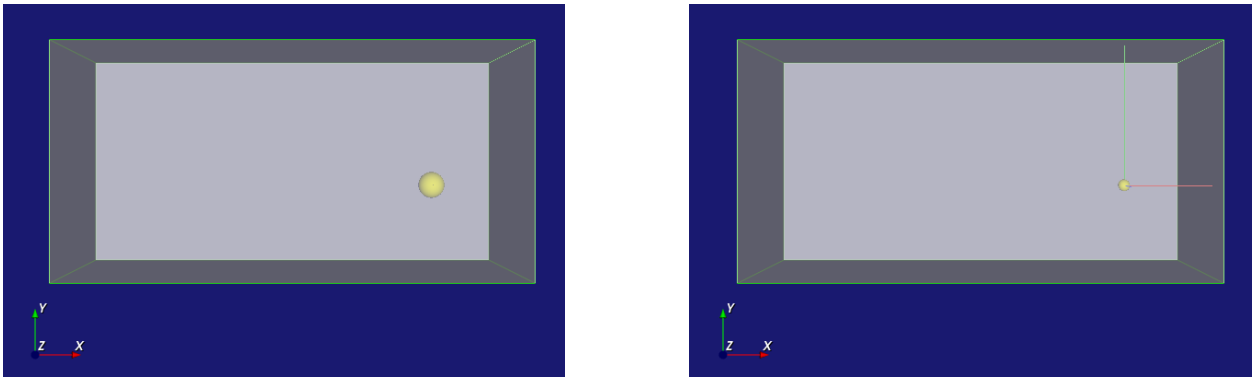
$$\mu_{new} = \frac{\sum_i w_i \boldsymbol{x_i}}{\sum_i w_i}$$

while the covariance Σ_{new} is calculated using

$$\Sigma_{new} = \frac{\sum_i w_i}{\left(\sum_i w_i\right)^2 - \sum_i w_i^2} \sum_i w_i (\boldsymbol{x_i} - \mu_{new})^T (\boldsymbol{x_i} - \mu_{new})$$

where $\frac{\sum_i w_i}{\left(\sum_i w_i\right)^2 - \sum_i w_i^2}$ is the correction factor to get an unbiased estimate of the sample covariance matrix.

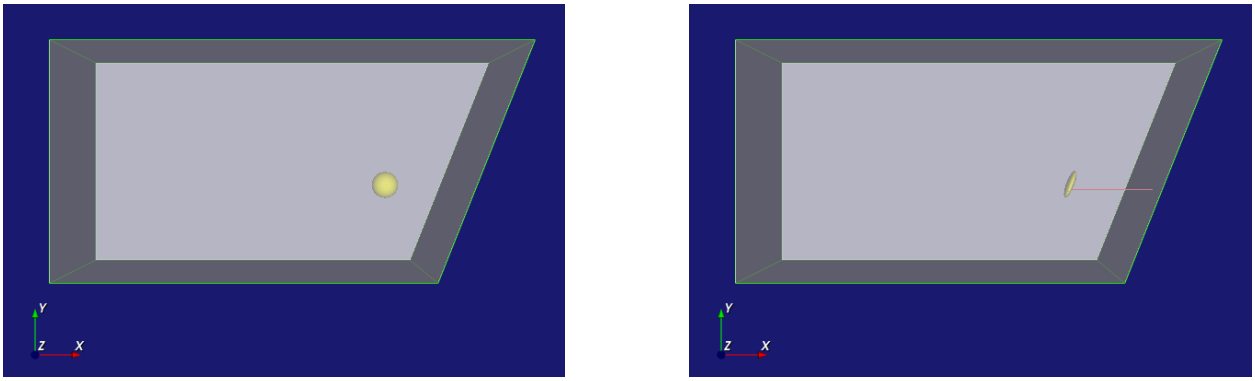
An example of a state update for a three pointlaser setup is shown in Figure 3.6. As expected, a measurement forces the state to converge towards the ground truth position.



(a) Initial State (b) Updated state after taking measurement

Figure 3.6: A general example of state update after a measurement

In the alternate case when there is only a single pointlaser, an example of the state update is shown below in Figure 3.7. A single pointlaser only provides information in one direction, and this is exactly what we observe in the updated state.



(a) Initial State (b) Updated state after taking measurement

Figure 3.7: An example of state update when there is only a single pointlaser

CHAPTER 4
PROBLEM FORMULATION

We formulate our active sensing objective as a sequential decision making problem. At the outset, the position belief is spread over a large volume in space. The decision making problem is to figure out the optimal sequence of orientations for taking measurements, which lead to the maximum reduction in the position belief uncertainty.

This decision process is modelled as a fully-observable Markov Decision Process (MDP). At any instant, the agent has perfect information of it's state s_t, which it can use to take an action a_t. This action results in a transition to a new state s_{t+1} and is accompanied by a reward r_t. The purpose of the agent is take the sequence of actions (called a policy π) which results in the maximum cumulative reward. More concretely, our MDP involves the following set of signals:

1. Observation (o_t)

   The observation is defined as the position belief of the robot. This is parameterised using the state description X from Section 3.1. Further, we also evaluate agents which additionally store a history of their actions in the observation.

2. Action (a_t)

   The action is defined as an absolute orientation of the pointlaser array from which to take a new measurement.

3. Reward (r_t)

   The immediate reward at every step is the reduction in uncertainty (or the information gain) of the position belief after taking a new measurement.

Hereafter, we use the observation and state interchangeably since the MDP is formulated to be fully-observational and thus, an observation provides complete information of the

underlying state. The interactions between these signals has been visualised in Figure 4.1.

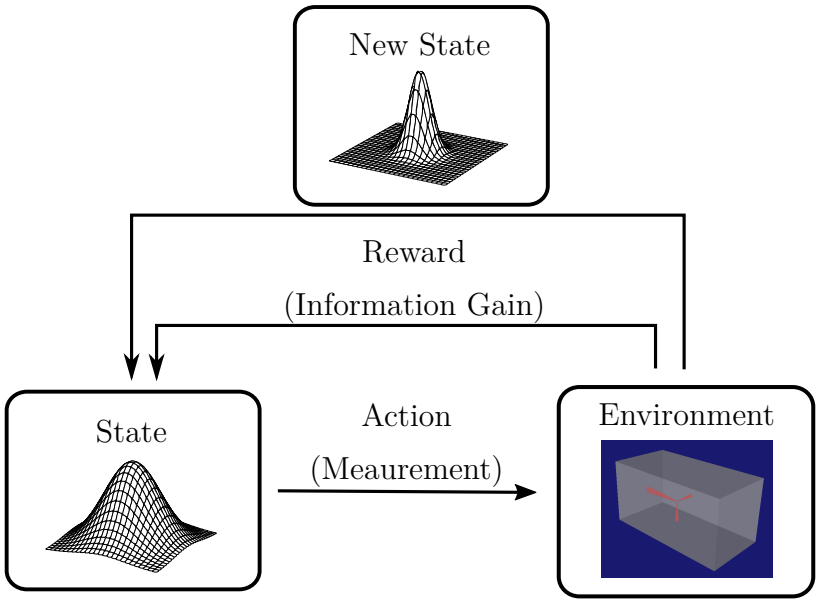


Figure 4.1: Interactions involved in our uncertainty reduction MDP

We hypothesise that, for a particular mesh, these signals provide the agent with enough information to make optimal choices for reducing the uncertainty in position. These signals have been further discussed in the following sections.

4.1 Observation Space

For our purposes, the observation space is specific to a particular mesh. The space contains the belief of the robot position, which is represented as a trivariate gaussian distribution parameterised by a mean $\mu = (\mu_x,\ \mu_y,\ \mu_z)$, and unique values in the covariance matrix $(\Sigma_{xx},\ \Sigma_{xy},\ \Sigma_{xz},\ \Sigma_{yy},\ \Sigma_{yz},\ \Sigma_{zz})$. This space is continuous and the values are normalised to ensure that they remain bounded within a fixed interval.

During the course of experimentation, we discovered that augmenting these values with a history of actions makes it easier for the agent to learn optimal decisions. Hence, we expand the observation space to include a limited history of previous actions. More details regarding this have been reserved for the next chapter.

4.2 Action Space

Based on the observation, the agent decides to take an action that is defined as the next best absolute orientation from which to take a measurement. This orientation is 'absolute' in the sense that it represents a rigid body rotation relative to a fixed coordinate system at the origin, and not relative to a previous pose. Thus, the action space encompasses all elements of the $SO(3)$ rotation group and hence, is also continuous.

This rigid body rotation can be mathematically represented in a number of ways:

1. Euler Angles $(\alpha,\ \beta,\ \gamma)$

   They represent a sequence of three rotations, either as the rotation of the axes of the base coordinate system to the target rigid body system (extrinsic rotations) or vice-versa (intrinsic rotations). These can further be classified based on the order of rotations (eg. $x-y-z$, $z-y-z$, $x-z-y$ etc.).

   Here, $\alpha \in [-\pi, \pi]$, $\beta \in [-\frac{\pi}{2}, \frac{\pi}{2}]$ and $\gamma \in [-\pi, \pi]$.

2. Quaternion $(w,\ x,\ y,\ z)$

   Unit quaternions are a numerically stable alternative to Euler angles which represent a rotation as a generalised complex vector $w + x\mathbf{i} + y\mathbf{j} + z\mathbf{k}$ with a unit norm. They are usually stored as four values.

3. Rotation Matrix R

   They are full 3x3 transformation matrices which are used to transform any vector in the base coordinate frame to a vector in the target rigid body coordinate frame.

There is some literature suggests that Euler angles/unit quaternions are difficult to learn for neural networks as compared to higher dimensional representations [27] due to inherent discontinuities in their representation. Experimentally, we didn't find a significant difference in performance. Hence, we restrict ourselves to basic 3D Euler angles representation, and use $x-y-z$ sequence of extrinsic rotations.

4.3 Reward

Following an action, the agent interacts with the environment by taking a measurement from the new orientation. Based on this measurement, the state is updated as described in Section 3.3, thereby resulting in a new state with a reduced uncertainty. This reduction in the uncertainty U (or gain in information) is usually quantified using either entropy of the belief distribution, or through scalar functions of the covariance matrix Σ. For our trivariate gaussian belief, reducing entropy-based uncertainty is equivalent to reducing the determinant of the covariance matrix, and hence, is a subset of the latter category. The covariance matrix functions are frequently expressed as functions of its eigenvalues λ. These are intuitive to understand geometrically by visualising the covariance ellipsoid (Figure 3.2a). Three common examples of such functions are

1. Volume of uncertainty ellipsoid (D-optimal design)

$$U^{vol} = \sqrt[3]{\lambda_1 \lambda_2 \lambda_3} = \sqrt[3]{|\Sigma|}$$

2. Length of the major axis (E-optimal design)

$$U^{axis} = \max\{\lambda_1,\ \lambda_2,\ \lambda_3\}$$

3. Sum of lengths of ellipsoid axes (A-optimal design)

$$U^{sum} = \lambda_1\ +\ \lambda_2\ +\ \lambda_3 = \mathrm{tr}(\Sigma)$$

These functions hold the added advantage of having the same units as position, and therefore, their numerical values are easy to interpret. During the course of our experiments, a reward comprising of a combination of the volume and the major-axis length was found to be most suitable for learning.

4.4 Summary

Our overall decision making problem can be articulated as:
"Given a state s_t comprising of the position belief and a recent history of pointlaser orientations, find a policy π^* which outputs the optimal pointlaser orientation from where to take the next measurement."

The condition for optimality is the cumulative reduction in the covariance matrix based uncertainty. Here, the policy maps a state to an action i.e. $\pi : s_t \rightarrow a_t$ such that

$$a_t = [\alpha,\ \beta,\ \gamma]_t = \pi(\mu_t,\ \Sigma_t,\ a_{t-1} \dots)$$

In the coming chapter, we show how this policy can be approximated by a neural network and trained using deep reinforcement learning.

CHAPTER 5
DEEP REINFORCEMENT LEARNING

Reinforcement Learning (RL) is a framework used to optimise a sequential decision making problem. It consists of an agent which learns to act by trial and error through interaction with the environment. These interactions result in a feedback from the environment in the form of immediate reward received by the agent. The agent's objective is to learn actions which maximise it's cumulative reward (often called the return)

$$R = \sum_t \gamma^t r_t$$

where γ is the discount factor and r_t are the rewards at different timesteps.

During the learning process, the agent incrementally adjusts its behaviour to fulfil this objective i.e. it learns to take actions which result in a higher cumulative reward and avoids actions which lead to a lower cumulative reward.

Formally, the objective of RL is to find an optimal policy $\pi^* : s_t \rightarrow a_t$ that maximises the expected return

$$\pi^* = \underset{\pi}{\text{argmax}}\ \mathrm{E}[R]$$

where π is the set of all possible policies.

5.1 Neural Networks

Neural networks (Figure 5.1) are widely used as function approximators to tackle complex high dimensional problems. Conventional approaches to solve such problems usually require certain manually handcrafted features in order to achieve good results. On the other hand, neural networks trained end-to-end automatically extract useful features for the task-at-hand and provide superior performance. This circumvents the need for manually engineering

features, provided the neural network is exposed to sufficient representative data. Moreover, neural networks can also be scaled to higher dimensional input and output spaces without an exponential increase in the required training data [28].

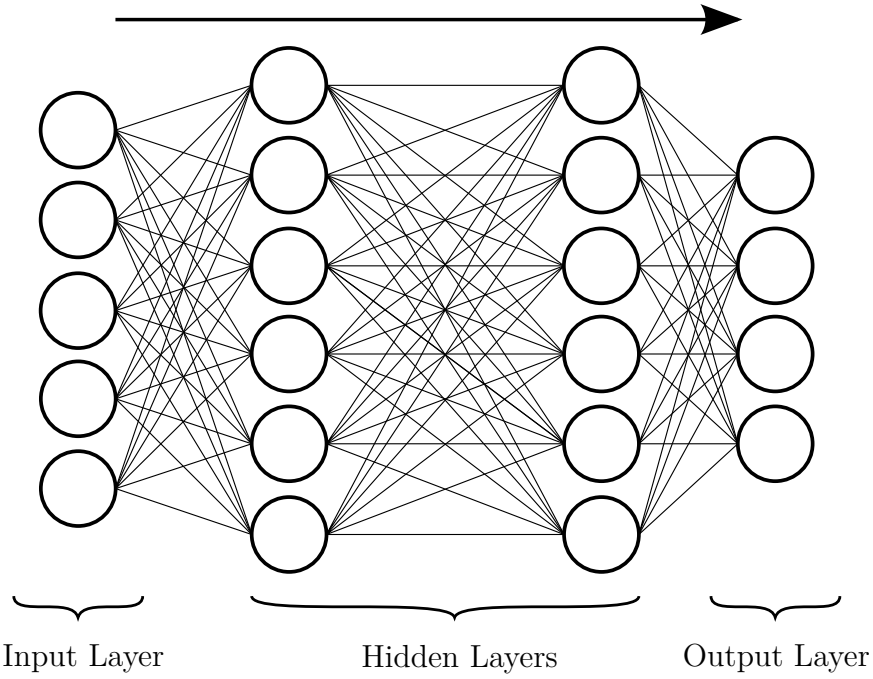


Figure 5.1: An example neural network consisting of two hidden layers

Essentially, a neural network is a non-linear function approximator f that maps an input x to an output y. It consists of a sequence of hidden layers containing neurons, where each layer performs a non-linear transformation of the inputs (referred to as activation). In addition, there is an input and output layer. The parameters of the neural network are the weights assigned to the connections between neurons of one layer and the next. Together, these weights are represented as θ.

The objective of training a neural network is to obtain a set of parameters such that value of a predefined loss function is minimised. This is done iteratively using gradient decent. At each iteration, the gradient of the loss is calculated with respect to each parameter through backpropagation, and this is used to update the weights.

Use of neural networks in conjunction with reinforcement learning is termed Deep Re-

inforcement Learning. When dealing with state and action spaces of high dimensions, it is often computationally intractable to use exact functions to represent the policy. Hence, neural networks are used to approximate the policy functions π_θ.

5.2 Policy Update

Our MDP formulation from Chapter 4 is made up of continuous observation and action spaces. Consequently, we resort to policy gradient methods [29] to solve this problem. Policy gradient methods are a subclass of RL algorithms that aim to directly infer the optimal policy π^*. Furthermore, we use a stochastic policy that maps the state to a probability distribution over possible actions. A diagonal multivariate gaussian is used to represent this distribution. The action is then sampled from the distribution i.e.

$$a_t \sim \pi_\theta(s_t)$$

In on-policy training, the agent learns the desired behaviour in a progressive manner. To begin with, the parameters of the policy are initialised randomly. At each iteration, the current policy is used to collect samples of agent's interactions with the environment for a specified number of episodes. Based on the rewards received from these episodes, the policy is updated. This process repeats for a fixed number of iterations or until the agent achieves the desired behaviour.

The policy update at each iteration is done by computing the gradient of the expected return of the policy (denoted as $J(\pi_\theta)$) with respect to the policy parameters θ. This gradient is given by

$$\nabla_\theta J(\pi_\theta) = \mathrm{E}_\tau \left[\sum_t \nabla_\theta \log \pi(a_t \mid s_t, \theta) R(\tau) \right]$$

where expectation is taken over an episode τ.

Vanilla policy gradient methods directly update the policy through gradient ascent using

the gradient calculated above, along with a suitable step size α

$$\theta_{t+1} = \theta_t + \alpha \nabla_\theta J(\pi_\theta)$$

This should ideally increase the probability of actions that lead to positive returns and reduce the probability of those that lead to lower returns. In addition to this, a critic may be used as part of the training process. This critic acts as a baseline for the returns. In this case, the gradient is given by

$$\nabla_\theta J(\pi_\theta) = \mathrm{E}_\tau \left[\sum_t \nabla_\theta \log \pi(a_t \mid s_t, \theta)(R(\tau) - B) \right]$$

where B is the baseline given by the critic.

A commonly used baseline is the estimate of the value function. The value function $V^\pi(s)$ is the expected return if the agent starts from the state s and acts according to the policy π i.e.

$$V^\pi(s) = \mathrm{E} \left[\sum_t \gamma^t r_t \right]$$

The difference between return received by following a set of actions from a given state s and the expected reward from that state is known as the advantage A

$$A = \sum_t \gamma^t r_t - V(s)$$

Intuitively, maximising the advantage would encourage the agent to take actions that lead to a higher than expected reward, and discourage those that lead to a lower than expected reward.

Although policy gradient methods can be applied without using value functions, the latter reduces the variance of gradients during updates. This makes the training more stable and ensures faster convergence.

5.3 Trust Region Policy Optimisation (TRPO)

Even with a critic, vanilla policy gradient methods can still have a significantly high variance in the calculated gradients. Even for small step sizes, this could possibly lead to bad policy updates from which it might be difficult for the agent to recover. In order to overcome this, newer methods were developed which restrict the policy changes at each iteration, while still guaranteeing improvements.

One such algorithm is TRPO which ensures that the policy update at each iteration is constrained around a certain trust region. The trust region is defined in terms of the average KL divergence between the action distributions before and after a policy update. Formally, the TRPO optimisation problem is written as

$$\begin{aligned} \underset{\theta}{\text{argmax}} \quad & J(\pi_\theta) \\ \text{s.t.} \quad & \bar{D}_{KL}(\theta, \theta_{old}) \leq \delta \end{aligned}$$

where δ is the desired KL divergence.

The KL divergence constraint certifies that the policy doesn't change erratically, thereby ensuring stable policy updates. Additionally, it also ensures that the policy doesn't overfit to the latest set of samples. Solution to the TRPO optimisation has been derived by the authors in the appendix of [30]. The final algorithm is summarised below

Algorithm 1: Trust Region Policy Optimisation

Initialise the policy with parameters θ_0 and value function with parameters ϕ_0.

for *i = 0, 1, 2 ...* **do**

- Run the policy π_{θ_i} for N timesteps
- Compute the advantages A^t using $V = V_{\phi_i}$ for all timesteps $t \in 1 \dots N$
- $A^t = \sum_l \gamma^l r_{t+l} - V(s_t)$
- Calculate θ_{i+1} using TRPO update
- Calculate ϕ_{i+1} using Trust region update

end

CHAPTER 6
EXPERIMENTS

To demonstrate the efficacy of our method, we train the RL agent with a sensor configuration consisting of a single pointlaser. In this case, it is expected that the agent would take measurements in such a way that the laser ray falls perpendicular to close walls. Also, we reckon that successive measurements would be taken from approximately orthogonal directions. We test this conjecture on a number of different meshes.

Before proceeding with the results, it is imperative to discuss a few important details.

6.1 Normalised observations and actions

We normalise the observation values in order to ensure that they are comparable across different meshes. The means μ_is are linearly mapped between $[0, 1]$ by making use of the respective bounding box coordinates of the mesh.

Normalising the Σ_{ij}s is not as straightforward. For the diagonal elements, we take the square root of the values, which represent the standard deviations. Further, we divide these with a maximum standard deviation value σ_{max}. As for the off-diagonal elements, we instead store the correlation values which are bounded between $[-1, 1]$.

Along with these, we also include the previous two actions in the observation. Since there is no history of actions at the start of an episode, we initialise these values with zeros. After the first action, three of these zeros are replaced by the normalised first action output (between $[-1, 1]$). This continues on as the episode progresses.

Therefore, the observation is a 15 tuple with the first 3 values representing the belief mean, next 6 representing the belief covariance, and the last 6 representing the previous two actions. This 15 dimensional observation is inputted to the policy neural network which outputs a distribution over the 3 dimensional action space.

The action represents a pointlaser array orientation and is sampled from a normalised 3D gaussian distribution. The mean of the distribution is the output of the neural network, while the three standard deviations are separate learnable parameters. The action values are clipped between $[-1, 1]$ and the orientation is obtained by denormalised these values according to the Euler angle range limits given in Section 4.2.

6.2 Crafted reward

After each step, the reward given to the agent is the reduction in uncertainty i.e. $-(U_t - U_{t-1})$. This uncertainty can be expressed in a number of ways as described in Section 4.3. We noticed that using only the determinant uncertainty (ellipsoid volume) results in the agent learning to reduce the uncertainty solely in one direction. Visually, the uncertainty ellipsoid is squeezed along a single axis, while the uncertainty in the other two axes remains constant. In order to mitigate this, we supplement the immediate rewards with a final episodic reward which is given by reduction in the length of the major axis of the uncertainty ellipsoid between the initial and final beliefs ($-[U^{axis}_{final} - U^{axis}_{initial}]$). This ensures that the uncertainty ellipsoid is squeezed in all directions.

6.3 Episode Length

In our experiments, we restrict an episode to 3 timesteps. On each timestep, a pointlaser orientation is used to take a measurement and the uncertainty of the updated belief is noted. At the end of an episode, the final belief uncertainty needs to be minimised.

6.4 Network Architecture and Training Details

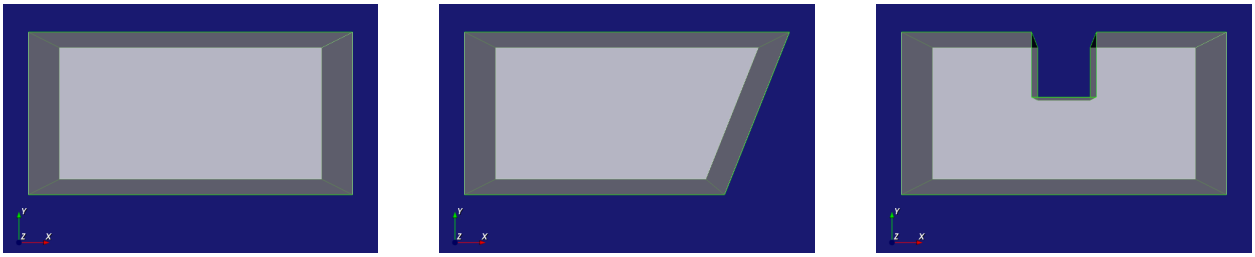
Our neural network consists of 2 layers, each containing 64 neurons. These layers are connected with tanh activations. The weights are initialised randomly from a standard normal distribution. Moreover, there are three additional learnable standard deviations which are

initialised as $[0.5,\ 0.5,\ 0.5]$.

The agent is trained for 750 iterations using TRPO. Each training iteration consists of data from 500 episodes. Along with the 3 timestep episode, this gives a training batch size of 1500. The training is done in Python using TensorFlow [31].

6.5 Results

We evaluate our method by training on three different sample meshes (Figure 6.1). Each of these mesh is bounded by a box of dimensions 10000 x 5000 x 5000.



(a) Mesh A (b) Mesh B (c) Mesh C

Figure 6.1: The three meshes used for training and evaluating the agent

At the beginning of each episode, the agent starts from a random position inside the mesh, and with an initial uncertainty covariance matrix $\begin{pmatrix} 2500 & 0 & 0 \\ 0 & 2500 & 0 \\ 0 & 0 & 2500 \end{pmatrix}$. This signifies a position standard deviation of 50 in each of the XYZ directions.

Based on the output of the policy network, the agent then takes successive pointlaser measurements from three different orientations and reduces the belief uncertainty.

6.5.1 Mesh A

The first mesh is a simple cuboid with its edges parallel to the coordinate axes. For this mesh, we anticipate that the ideal three measurements would be along the XYZ directions. The training curves for this mesh are shown in Figures 6.2 and 6.3.

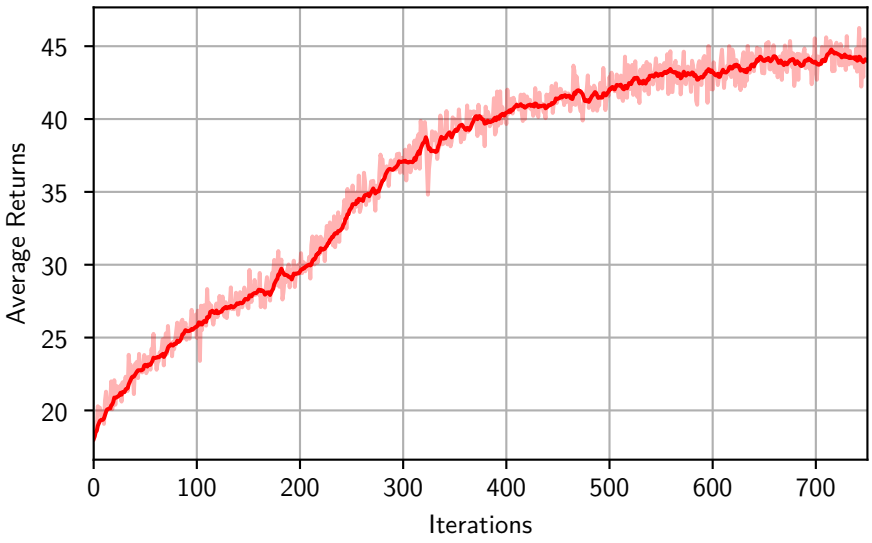
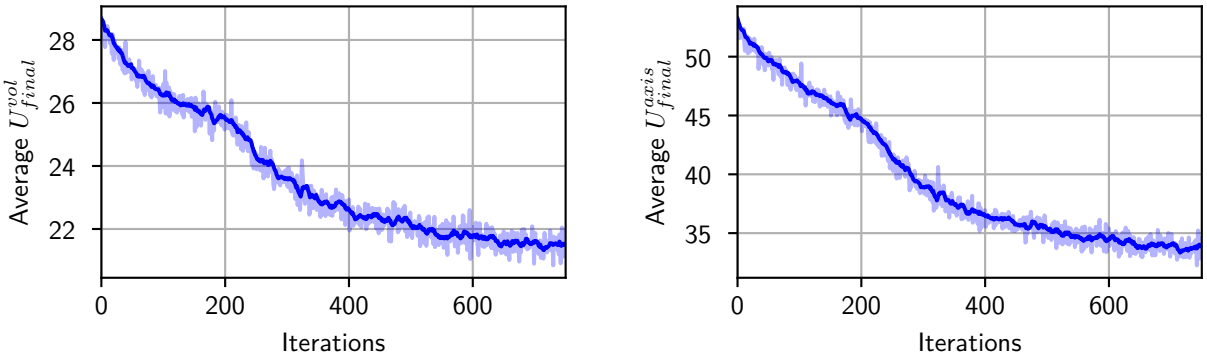


Figure 6.2: Learning curve of the average returns of an iteration for Mesh A

From Figure 6.2, it is observed that the cumulative reward continuously increases with the iterations until it saturates at a final value.



(a) Ellipsoid Volume

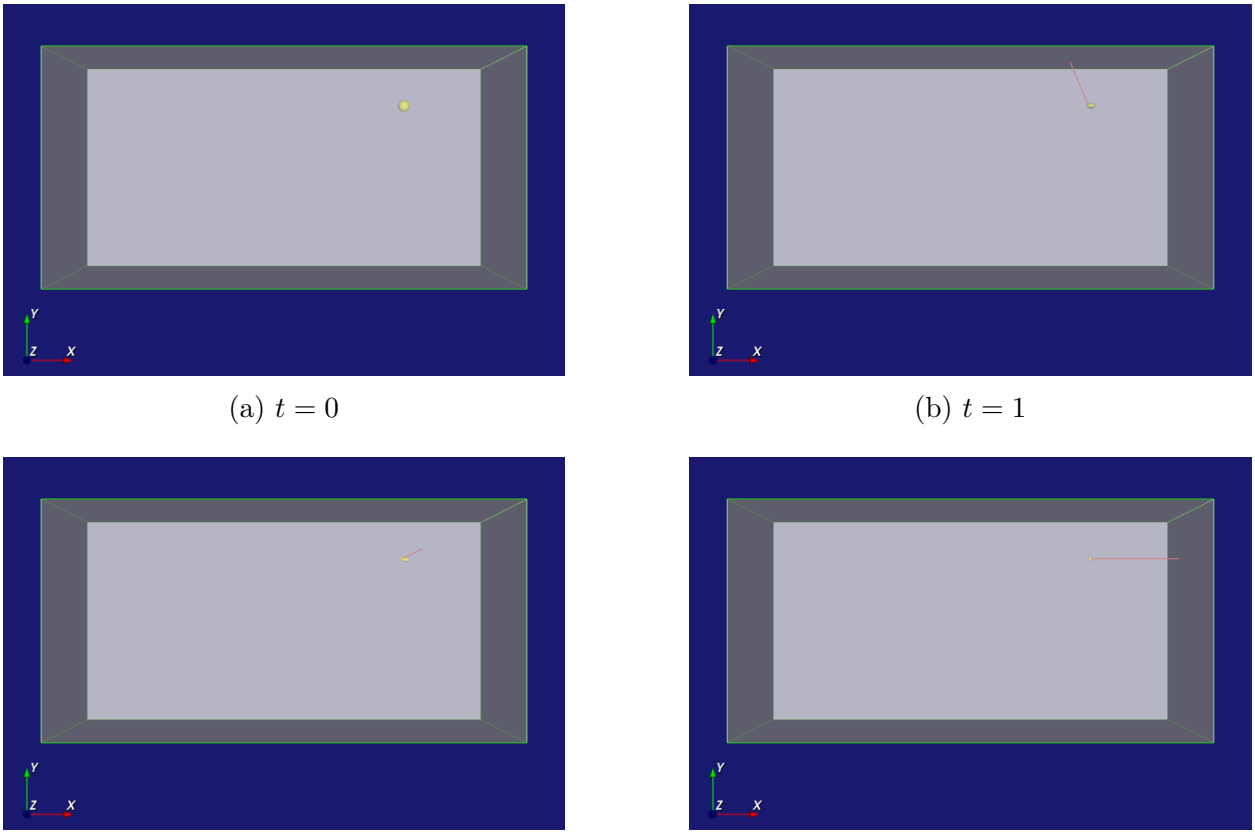
(b) Ellipsoid major axis length

Figure 6.3: Learning curves of two uncertainty metrics for Mesh A

Figure 6.3 shows that both the volume and major axis length of the uncertainty ellipsoid

reduce incrementally as the training progresses. The final uncertainty is significantly less than the initial value.

A qualitative assessment of the trained model shows that, in most cases, the final policy indeed orients the pointlaser approximately orthogonal to the walls (see Figures 6.4 and 6.5 for a couple of examples).



(a) $t = 0$ (b) $t = 1$

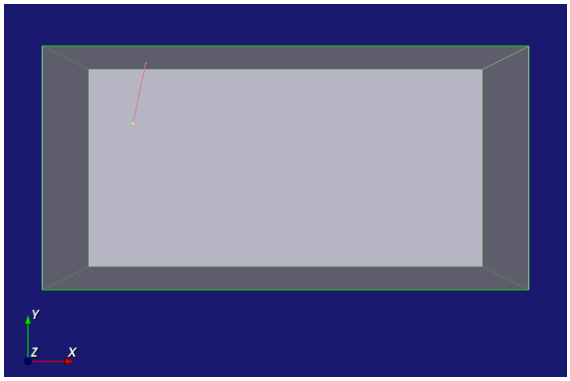
(c) $t = 2$ (d) $t = 3$

Figure 6.4: Rollout of the trained policy from state 1 in Mesh A

(a) $t = 0$

(b) $t = 1$

(c) $t = 2$



(d) $t = 3$

Figure 6.5: Rollout of the trained policy from state 2 in Mesh A

6.5.2 Mesh B

The second mesh has an oblique wall that makes it a little bit more complicated. The training curves for this mesh have been shown in Figures 6.6 and 6.7. We observe that the model converges for this mesh as well. Also, the average values of the final position uncertainty are comparable to the values obtained for Mesh A.

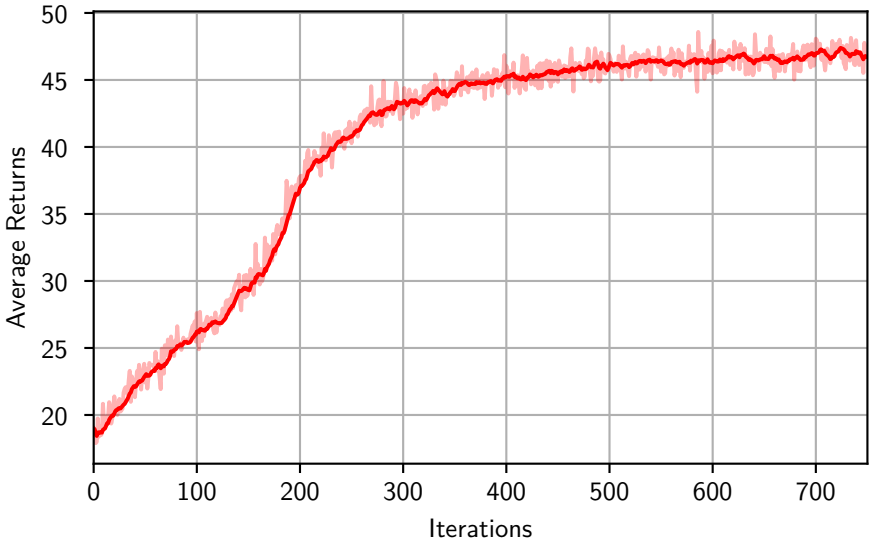
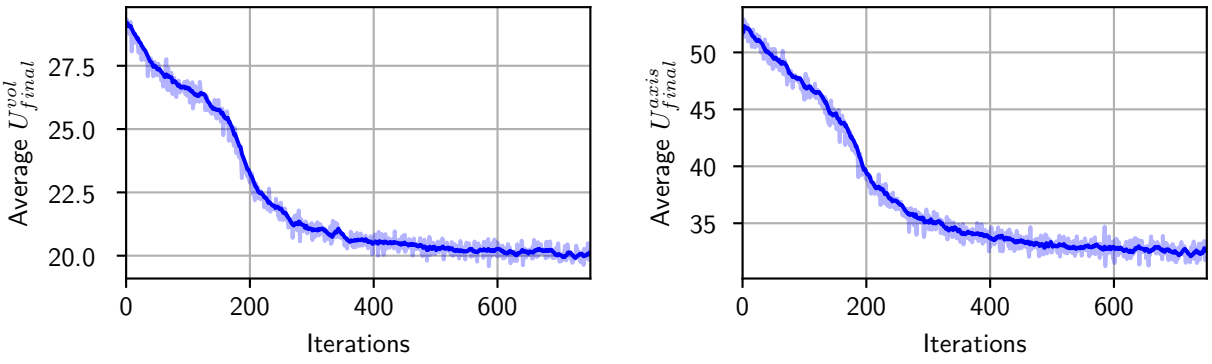


Figure 6.6: Learning curve of the average returns of an iteration for Mesh B

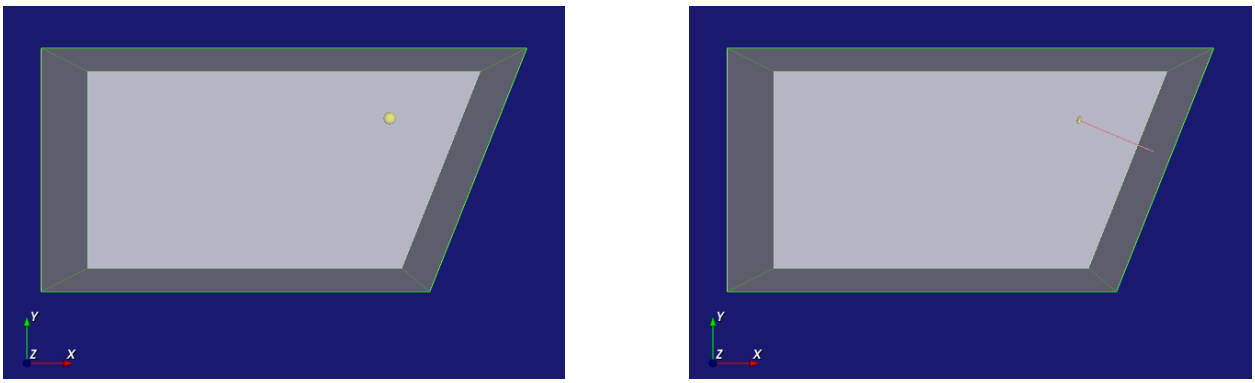


(a) Ellipsoid Volume

(b) Ellipsoid major axis length

Figure 6.7: Learning curves of two uncertainty metrics for Mesh B

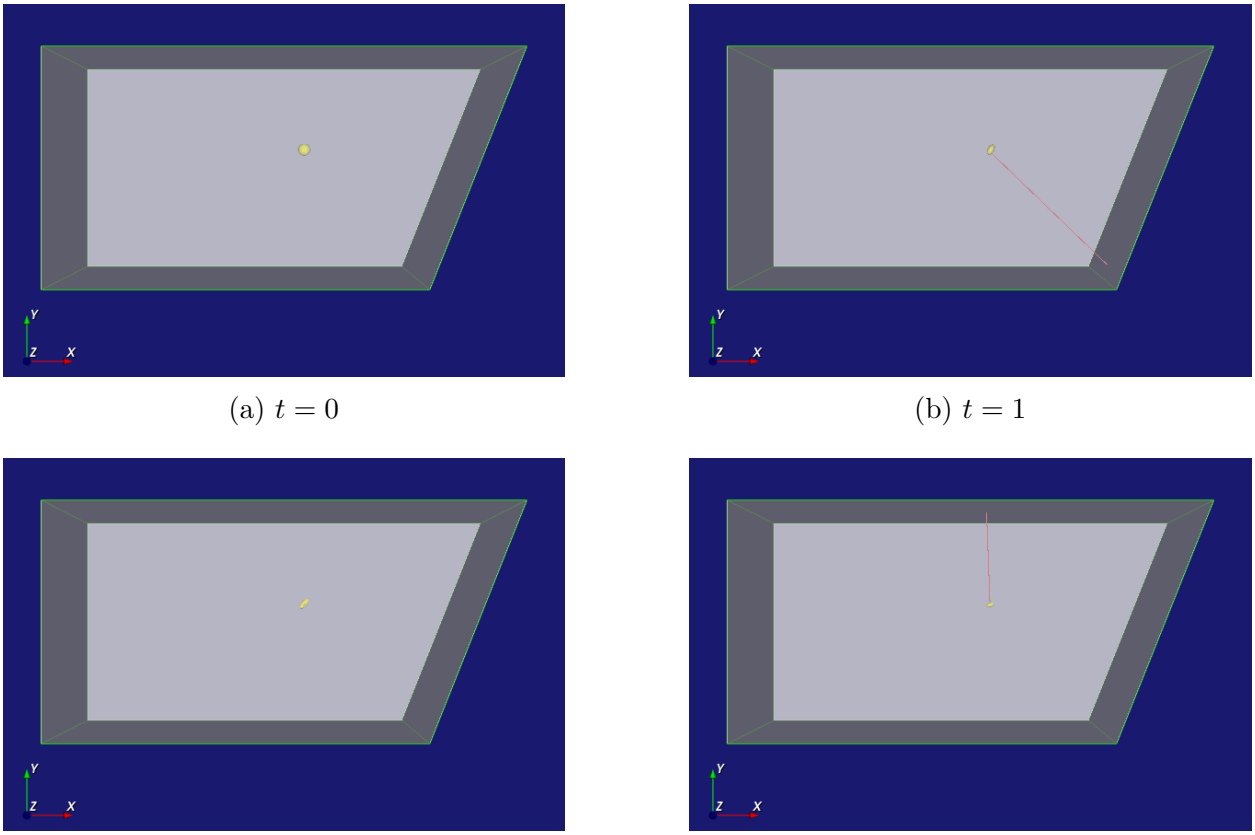
A couple of examples of the trained policy have been shown in Figures 6.8 and 6.9. We notice that the agent is able learn that measurements orthogonal to the oblique wall are more informative.



(a) $t = 0$ (b) $t = 1$

(c) $t = 2$ (d) $t = 3$

Figure 6.8: Rollout of the trained policy from state 1 in Mesh B



(a) $t = 0$ (b) $t = 1$

(c) $t = 2$ (d) $t = 3$

Figure 6.9: Rollout of the trained policy from state 2 in Mesh B

6.5.3 Mesh C

The third mesh is again a bit more complicated with a small volume carved out of the original cuboid. The training curves have been shown in Figures 6.10 and 6.11. Again, the model is able to converge with low average final position uncertainty values.

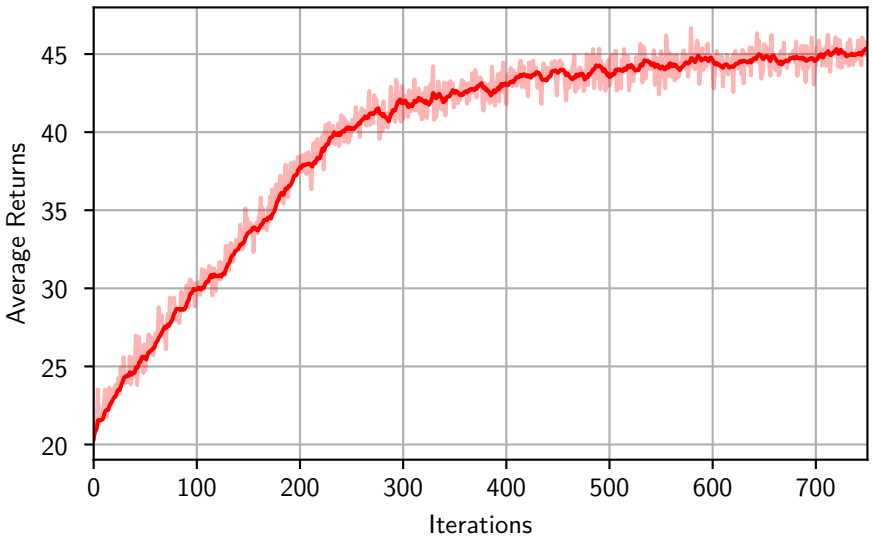
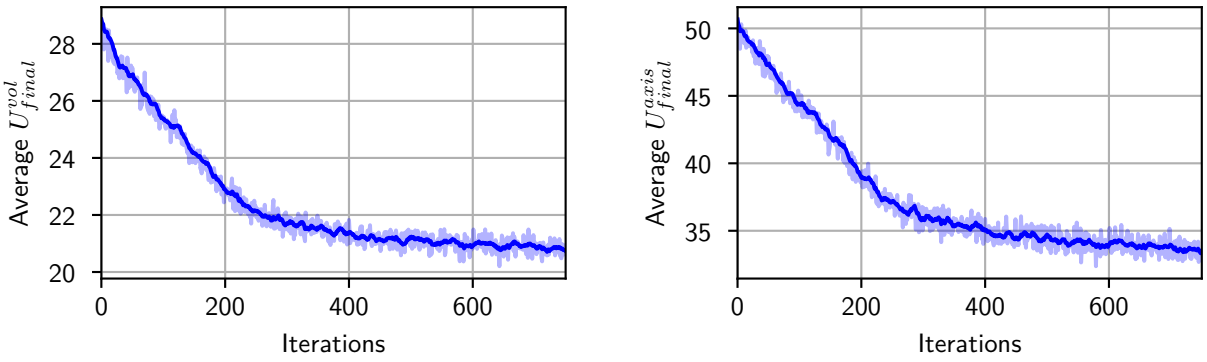


Figure 6.10: Learning curve of the average returns of an iteration for Mesh C

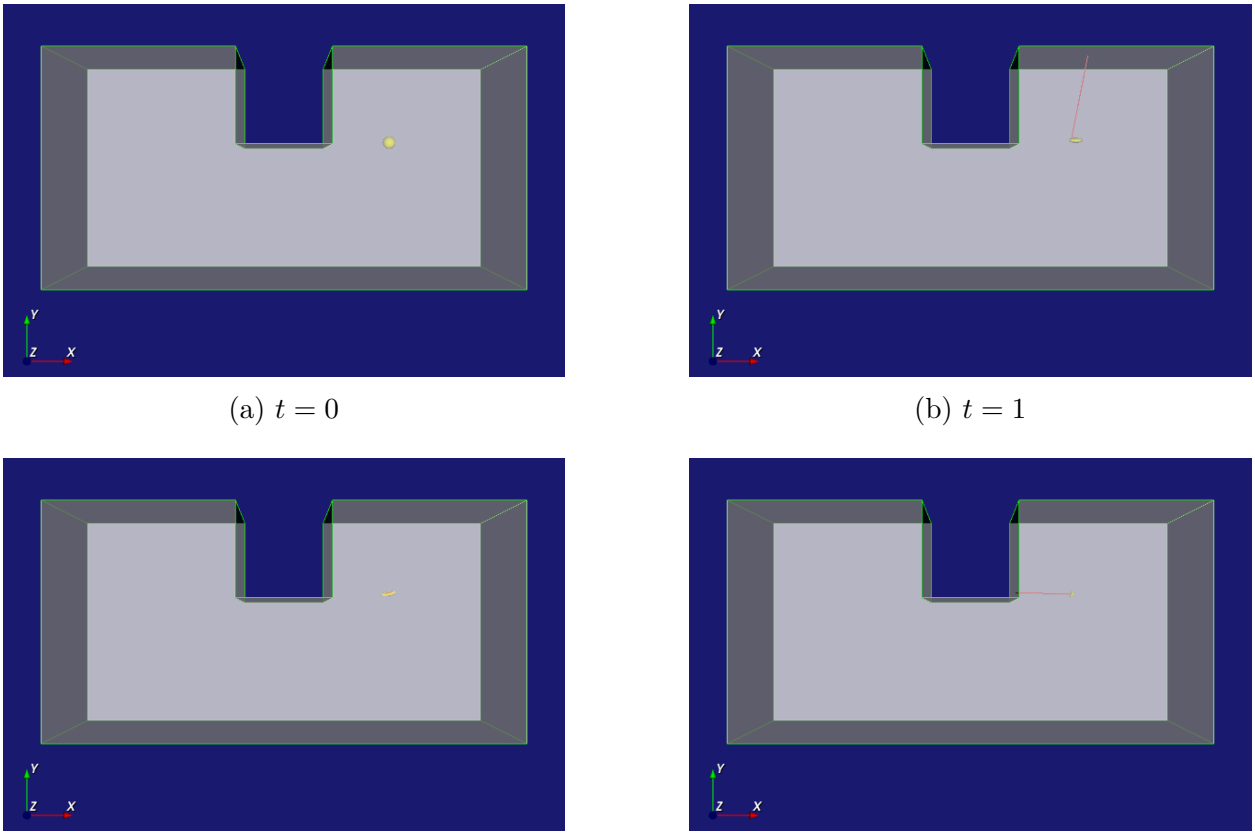


(a) Ellipsoid Volume

(b) Ellipsoid major axis length

Figure 6.11: Learning curves of two uncertainty metrics for Mesh C

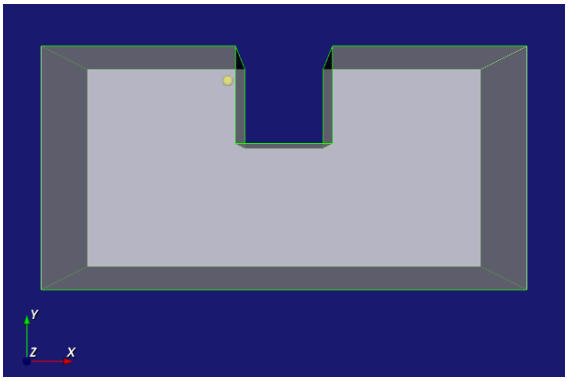
A couple of examples of the trained policy have been shown in Figures 6.12 and 6.13. We notice that the agent learns to take orthogonal measurements from walls which are closer.



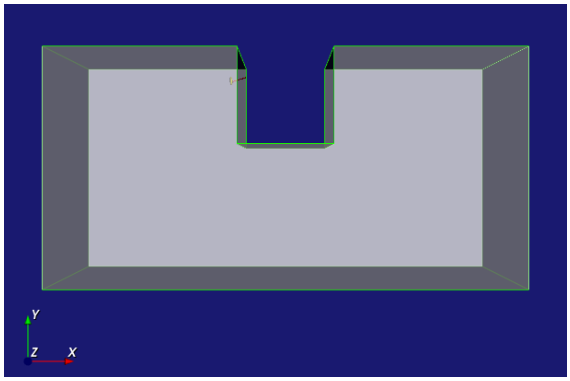
(a) $t = 0$ (b) $t = 1$

(c) $t = 2$ (d) $t = 3$

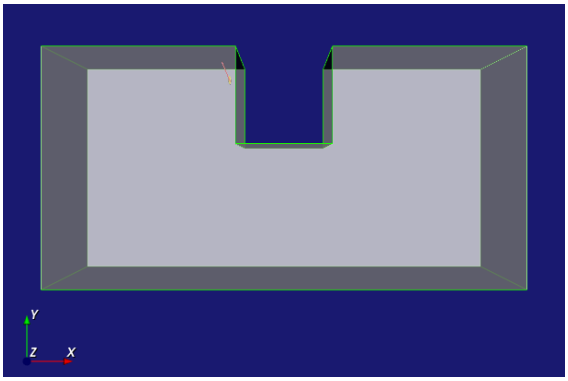
Figure 6.12: Rollout of the trained policy from state 1 in Mesh C



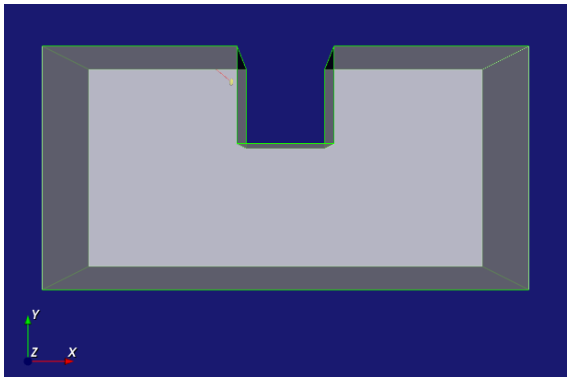
(a) $t = 0$



(b) $t = 1$



(c) $t = 2$



(d) $t = 3$

Figure 6.13: Rollout of the trained policy from state 2 in Mesh C

6.6 Comparison to a random policy

In order to establish that our method achieves good results, we compare the final uncertainties U_{final} obtained using our trained agent with the final uncertainties obtained from an agent which follows a random policy. This comparison is shown in Table 6.1.

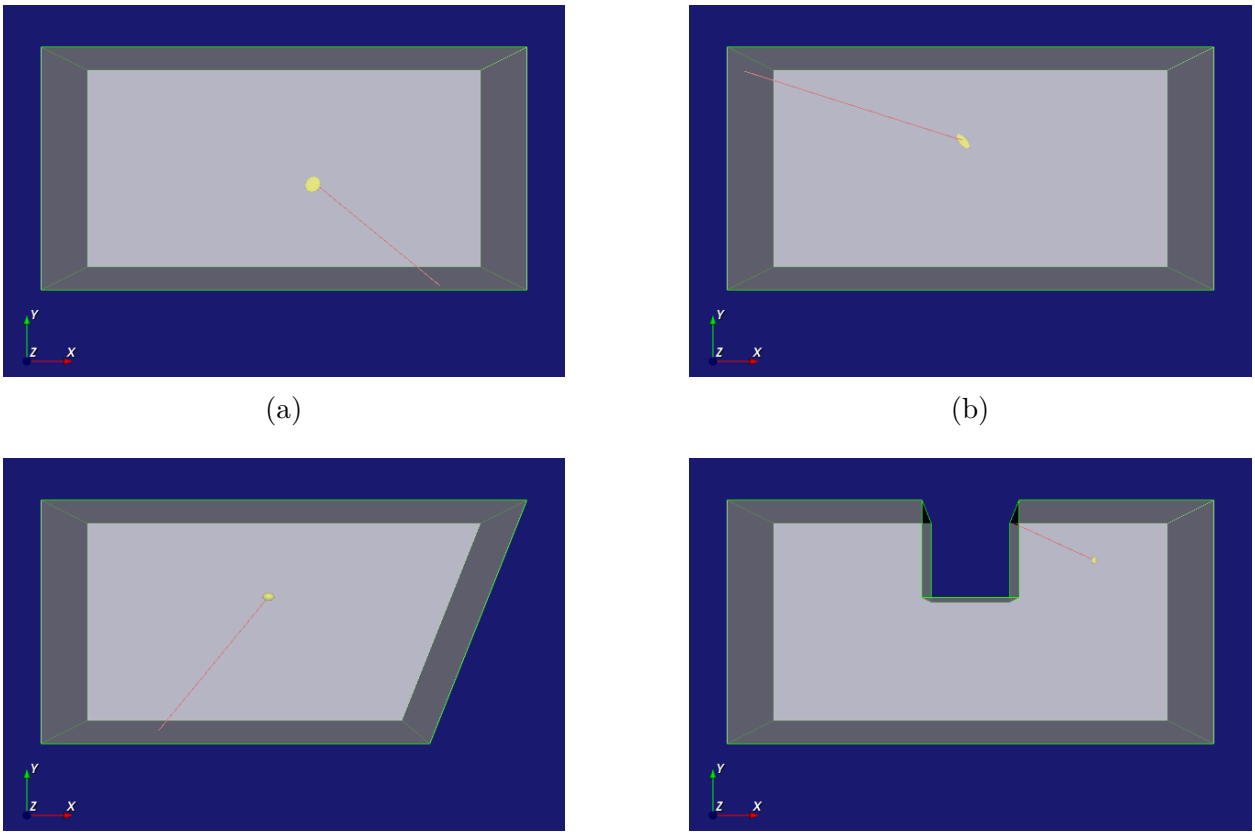
	U^{vol}_{final}			U^{axis}_{final}		
	A	B	C	A	B	C
Random Policy	29.79	29.37	28.68	51.08	51.53	49.29
TRPO	21.46	20.11	20.62	33.84	31.78	33.28

Table 6.1: Comparison of the average final uncertainties of two separate policy rollouts in the three sample meshes A, B and C.

We observe that the trained models perform considerably better than a random policy. Both the types of uncertainties start with an initial value $U_0 = U^{axis}_0 = U^{vol}_0 = 50$. While U_{axis} doesn't change at all throughout the episode for the random policy, it is interesting to note that the volume uncertainty U^{vol}_{final} undergoes a significant reduction. This further justifies our choice of the reward function where we included the final major axis length in the episode termination reward.

6.7 Failure Cases

The trained models are far from foolproof. Some failures have been shown in Figure 6.14.



(a) (b)

(c) (d)

Figure 6.14: States at which the trained agent predicts suboptimal actions

One of the reasons for these shortcomings is the uni-modal representation of the policy distribution. Such a representation impairs the ability of the agent to accurately predict actions at points where the optimal action changes abruptly with a small shift in position.

CHAPTER 7
CONCLUSION

In this thesis, we showed how reinforcement learning can be used to solve an active localisation problem of reducing the position uncertainty. The training was carried out on a small set of test meshes. We demonstrated that the RL agent is able to converge to approximately optimal solutions using our covariance matrix based information gain metric as the reward. It was observed that the policy does not perform well at boundaries where the optimal output is expected to change abruptly. This was attributed to our uni-modal gaussian representation of the stochastic policy. Overall, we believe that such simulations can be used for identifying good spots for taking measurements in CAD environments where the sensors involved are pointlasers.

The major limitation of our method is that we had to train the agent separately for different meshes because of restrictions in the state representation. In future work, it is critical to integrate mesh structure into the state to ensure that the model is able to generalise to other meshes. This is not trivial since common mesh data representations cannot be easily fed into neural networks. One way to avoid this is to use a local spherical/panoramic depth image of the mesh around a position, instead of absolute position coordinates. Another way could be to use an auto-encoder on a sampled point cloud of the mesh.

A further avenue of research is to experiment with other reinforcement learning algorithms that would make training more sample efficient. As an example, one could use off-policy methods like Deep Deterministic Policy Gradients (DDPG). This could potentially lead to substantially shorter training times.

Appendices

APPENDIX A

SAMPLING QUATERNIONS INSIDE A SPECIFIED CONE

In this chapter, we describe a method to randomly sample quaternions which lie within a small cone (henceforth described as 'noise-cone') around a given orientation (Figure A.1). Let $\boldsymbol{q} = [x,\ y,\ z,\ w]$ be a unit quaternion representing the given orientation with respect to a fixed coordinate system. The half-angle of the noise-cone is denoted as ϕ.

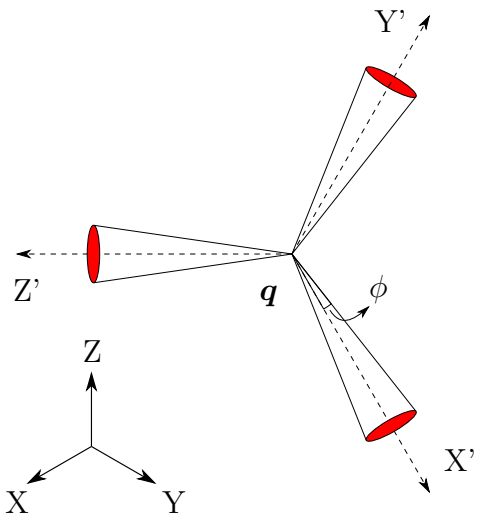


Figure A.1: Conical distribution of possible quaternion samples

Before proceeding, it is necessary to explain two important 3D geometry concepts which would be used later.

A.1 Angular distance between quaternions

For two quaternions, the angular distance θ between them is defined as the rotation angle required to transform one quaternion to the other. Let $\boldsymbol{q_1}$ and $\boldsymbol{q_2}$ be two unit quaternions.

The angular distance between them is given by

$$\theta = \cos^{-1}\left(2\langle \boldsymbol{q_1}, \boldsymbol{q_2}\rangle^2 - 1\right)$$

where $\langle \boldsymbol{q_1}, \boldsymbol{q_2}\rangle = x_1x_2 + y_1y_2 + z_1z_2 + w_1w_2$ is the quaternion inner product. The above equation can be rewritten as

$$\begin{aligned}\langle \boldsymbol{q_1}, \boldsymbol{q_2}\rangle &= \sqrt{\frac{1+\cos\theta}{2}} \\ &= \cos\frac{\theta}{2}\end{aligned}$$

A.2 Uniformly sampling a random unit vector

The problem of uniformly sampling a random unit vector is equivalent to uniformly sampling a point on the surface of a unit sphere. The solution to the latter is easy to visualise geometrically (Figure A.2).

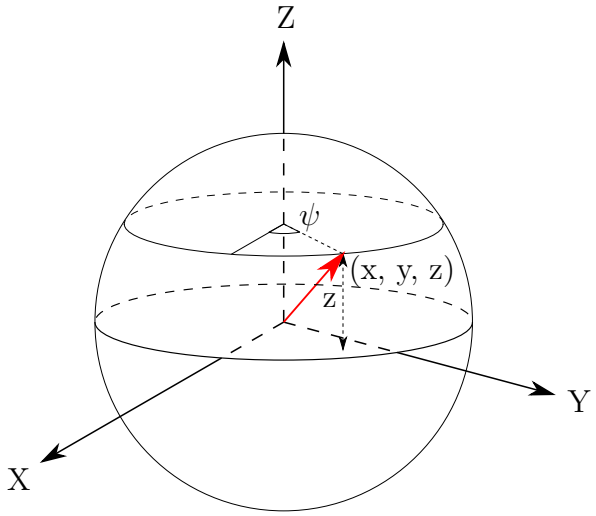


Figure A.2: Vector to point sampled uniformly over the surface of a unit sphere

Here, $z \sim \mathcal{U}(-1,\ 1)$ and $\psi \sim \mathcal{U}(0,\ 2\pi)$. By taking

$$x = \sqrt{1 - z^2} \cos \psi$$

$$y = \sqrt{1 - z^2} \sin \psi$$

we get the vector $\boldsymbol{v} = [x,\ y,\ z]$ which is a unit vector sampled uniformly over all the possible directions.

A.3 Sampling quaternions inside the noise-cone

Instead of directly sampling a quaternion $\boldsymbol{q_s}$ around the given orientation $\boldsymbol{q}$, we first generate a noise quaternion $\boldsymbol{q_n}$ around the identity $\boldsymbol{q_e} = [0,\ 0,\ 0,\ 1]$. Using the result from A.1, we have

$$\langle \boldsymbol{q_n}, \boldsymbol{q_e} \rangle = w_n = \cos \frac{\theta}{2}$$

By sampling θ from a uniform distribution over the noise-cone half-angle i.e. $\theta \sim \mathcal{U}(0,\ \phi)$, we can get samples for w_n. Further, by constraining $\boldsymbol{q_n}$ to be a unit quaternion, $(x_n,\ y_n,\ z_n)$ can be sampled from a sphere of radius $R = \sqrt{1 - w_n^2} = \sin \frac{\theta}{2}$ by scaling samples obtained from the method described in Section A.2.

The required sample $\boldsymbol{q_s}$ can finally be obtained by rotating the given quaternion $\boldsymbol{q}$ by the sampled noise $\boldsymbol{q_n}$ i.e. $\boldsymbol{q_s} = \boldsymbol{q} \cdot \boldsymbol{q_n}$.

APPENDIX B

CHARACTERISTICS OF POINTLASER MEASUREMENT NOISE

To approximate the measurement noise, we assume a simplified model of a pointlaser striking on a flat wall in 2D (Figure B.1). The pointlaser has an expected orientation and a small associated orientation noise ϕ. We take d as the distance to the wall, measured along the expected pointlaser orientation. In contrast, l is a random variable corresponding to the distance measurement along a noisy orientation. θ is the angle of incidence of the laser beam along the expected orientation.

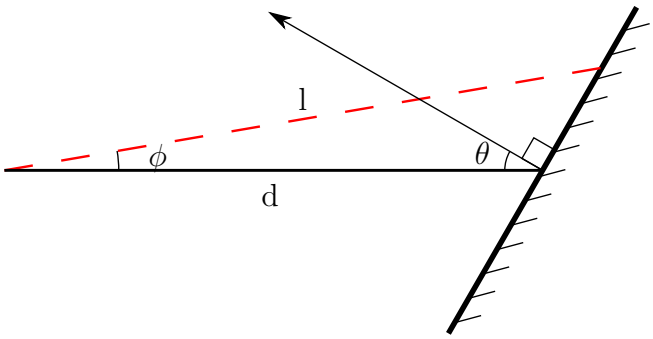


Figure B.1: Simplified 2D model of a pointlaser incident on a flat wall

Using sine rule in the enclosed triangle,

$$\frac{d}{\sin(\frac{\pi}{2} - \theta - \phi)} = \frac{l}{\sin(\frac{\pi}{2} + \theta)}$$

Solving for l, we get

$$\begin{aligned} l &= \frac{d\cos\theta}{\cos(\theta+\phi)} = \frac{d\cos\theta}{\cos\theta\cos\phi - \sin\theta\sin\phi} \\ &= \frac{d\sec\theta}{1 - \tan\theta\tan\phi} \end{aligned}$$

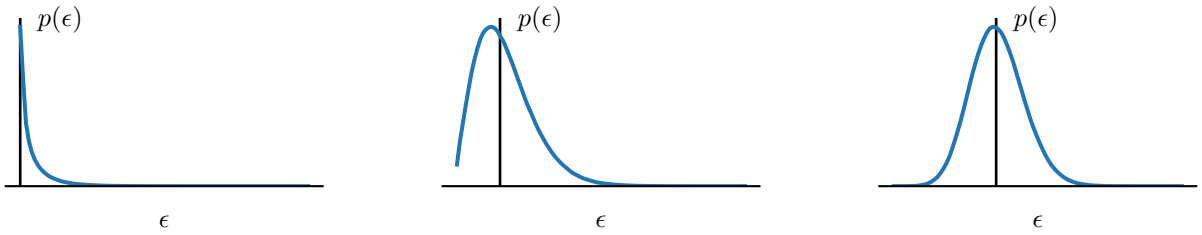
Since the orientation noise ϕ is assumed to be small,

$$\begin{aligned} l &\approx d \sec\phi(1 + \tan\theta \tan\phi) \\ &\approx d\left(1 + \phi\tan\theta + \frac{\phi^2}{2}\right) \end{aligned}$$

Now, noise in the distance measurement, $\epsilon = l - d$

$$\begin{aligned} \epsilon &\approx d\left(\phi\tan\theta + \frac{\phi^2}{2}\right) \\ &\propto d, \text{ and nonlinear in } \theta \text{ and } \phi. \end{aligned}$$

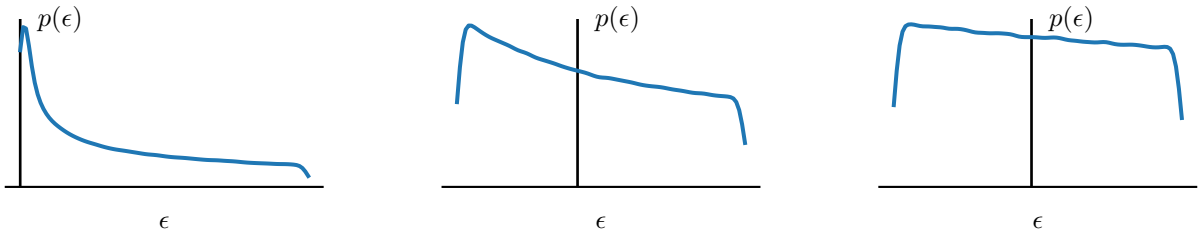
For a fixed d and a normally distributed ϕ, this noise has an exponential distribution for small θ, while it approaches a gaussian distribution as θ increases (Figure B.2).



(a) $\theta = 0$ (b) $\theta = \frac{\pi}{12}$ (c) $\theta = \frac{\pi}{4}$

Figure B.2: Variation of measurement noise PDF with angle of incidence $\left(\phi \sim \mathcal{N}\left(0, \frac{\pi}{36}^2\right)\right)$

Similarly, for a uniformly distributed ϕ, the noise distribution is shown in Figure B.3.



(a) $\theta = 0$ (b) $\theta = \frac{\pi}{12}$ (c) $\theta = \frac{\pi}{4}$

Figure B.3: Variation of measurement noise PDF with angle of incidence $\left(\phi \sim \mathcal{U}\left(-\frac{\pi}{36}, \frac{\pi}{36}\right)\right)$

REFERENCES

[1] A. Gawel, H. Blum, J. Pankert, K. Krämer, L. Bartolomei, S. Ercan, F. Farshidian, M. Chli, F. Gramazio, R. Siegwart, M. Hutter, and T. Sandy, "A Fully-Integrated Sensing and Control System for High-Accuracy Mobile Robotic Building Construction," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, to appear.

[2] H Ardiny, S Witwicki, and F Mondada, "Are autonomous mobile robots able to take over construction? a review," *International Journal of Robotics, Theory and Applications*, vol. 4, no. 3, pp. 10–21, 2015.

[3] T. Sandy, "High-accuracy mobile manipulation for on-site robotic building construction," PhD thesis, ETH Zurich, 2018.

[4] M. Allwright, N. Bhalla, H. El-faham, A. Antoun, C. Pinciroli, and M. Dorigo, "Srocs: Leveraging stigmergy on a multi-robot construction platform for unknown environments," in *International Conference on Swarm Intelligence*, Springer, 2014, pp. 158–169.

[5] M. Giftthaler, T. Sandy, K. Dörfler, I. Brooks, M. Buckingham, G. Rey, M. Kohler, F. Gramazio, and J. Buchli, "Mobile robotic fabrication at 1:1 scale: The in situ fabricator: System, experiences and current developments," *Construction Robotics*, vol. 1, no. 1-4, pp. 3–14, Dec. 2017.

[6] E Gambao, C Balaguer, and F Gebhart, "Robot assembly system for computer-integrated construction," *Automation in Construction*, vol. 9, no. 5, pp. 479–487, Sep. 2000.

[7] V. Lertpiriyasuwat and M. C. Berg, "Adaptive real-time estimation of end-effector position and orientation using precise measurements of end-effector position," *IEEE/ASME Transactions on Mechatronics*, vol. 11, no. 3, pp. 304–319, Jun. 2006.

[8] L. Stadelmann, T. Sandy, A. Thoma, and J. Buchli, "End-effector pose correction for versatile large-scale multi-robotic systems," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 546–553, Apr. 2019.

[9] K. Dörfler, T. Sandy, M. Giftthaler, F. Gramazio, M. Kohler, and J. Buchli, "Mobile robotic brickwork," in *Robotic Fabrication in Architecture, Art and Design 2016*, pp. 204–217.

[10] L. Mihaylova, T. Lefebvre, H. Bruyninckx, K. Gadeyne, and J. D. Schutter, "Active Sensing for Robotics – A Survey," 2002.

[11] W. Burgard, D. Fox, and S. Thrun, "Active mobile robot localization," in *Proceedings of the Fifteenth International Joint Conference on Artifical Intelligence - Volume 2*, 1997, pp. 1346–1352.

[12] D. Fox, W. Burgard, and S. Thrun, "Active markov localization for mobile robots," *Robotics and Autonomous Systems*, vol. 25, no. 3, pp. 195–207, Nov. 1998.

[13] Z. Zhang and D. Scaramuzza, "Beyond point clouds: Fisher information field for active visual localization," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2019.

[14] N. Roy, W. Burgard, D. Fox, and S. Thrun, "Coastal navigation-mobile robot navigation with uncertainty in dynamic environments," in *Proceedings 1999 IEEE International Conference on Robotics and Automation*, vol. 1, May 1999, pp. 35–40.

[15] S. M. LaValle, *Planning Algorithms*. Cambridge: Cambridge University Press, 2006.

[16] S. Thrun, W. Burgard, and D. Fox, *Probabilistic robotics*, ser. Intelligent robotics and autonomous agents. Cambridge, Mass: MIT Press, 2005, 647 pp.

[17] D. S. Chaplot, E. Parisotto, and R. Salakhutdinov, "Active neural localization," in *International Conference on Learning Representations*, 2018.

[18] S. Krishna, K. Seo, D. Bhatt, V. Mai, K. Murthy, and L. Paull, "Deep active localization," *arXiv preprint*, Mar. 5, 2019. arXiv: 1903.01669.

[19] C. Papachristos, M. Kamel, M. Popović, S. Khattak, A. Bircher, H. Oleynikova, T. Dang, F. Mascarich, K. Alexis, and R. Siegwart, "Autonomous exploration and inspection path planning for aerial robots using the robot operating system," in *Robot Operating System (ROS)*, Springer, 2019, pp. 67–111.

[20] S. Chen, Y. Li, and N. M. Kwok, "Active vision in robotic systems: A survey of recent developments," *The International Journal of Robotics Research*, vol. 30, no. 11, pp. 1343–1377, Sep. 2011.

[21] D. Falanga, P. Foehn, P. Lu, and D. Scaramuzza, "PAMPC: Perception-aware model predictive control for quadrotors," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Madrid: IEEE, Oct. 2018, pp. 1–8.

[22] J. Lundell, F. Verdoja, and V. Kyrki, "Hallucinating robots: Inferring Obstacle Distances from Partial Laser Measurements," *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4781–4787, Oct. 2018.

[23] J. Lundell, F. Verdoja, and V. Kyrki, "Deep network uncertainty maps for indoor navigation," *arXiv preprint*, Sep. 13, 2018. arXiv: 1809.04891.

[24] OpenAI, M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, J. Schneider, S. Sidor, J. Tobin, P. Welinder, L. Weng, and W. Zaremba, "Learning dexterous in-hand manipulation," *arXiv:1808.00177 [cs, stat]*, Jan. 18, 2019. arXiv: 1808.00177.

[25] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke, and S. Levine, "QT-opt: Scalable deep reinforcement learning for vision-based robotic manipulation," *arXiv:1806.10293 [cs, stat]*, Nov. 27, 2018. arXiv: 1806.10293.

[26] D. Stutz, "Learning shape completion from bounding boxes with CAD shape priors," p. 143,

[27] Y. Zhou, C. Barnes, J. Lu, J. Yang, and H. Li, "On the continuity of rotation representations in neural networks," *arXiv:1812.07035 [cs, stat]*, Apr. 11, 2019, 12. arXiv: 1812.07035.

[28] V. François-Lavet, P. Henderson, R. Islam, M. G. Bellemare, J. Pineau, *et al.*, "An introduction to deep reinforcement learning," *Foundations and Trends® in Machine Learning*, vol. 11, no. 3-4, pp. 219–354, 2018.

[29] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," in *Reinforcement Learning*, Springer, 1992, pp. 5–32.

[30] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International Conference on Machine Learning*, 2015, pp. 1889–1897.

[31] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.